Manchester Computing
Supercomputing, Visualization & eScience

ParaFEM Library

A Suite of Finite Element Analysis Codes

Lee Margetts, Mike Pettipher, Ian Smith
(lee.margetts@man.ac.uk,
m.pettipher@man.ac.uk,
Ian.smith@man.ac.uk)

19th May 2004
Cray User Group
Knoxville, Tennessee

THE UNIVERSITY of MANCHESTER

Acknowledgements

- Cray Inc. for access to Cray systems.
- Andrew Jones at University of Manchester for running the codes.

FEA on HPC

- FEA typically not one of the major users of HPC.
- In UK, neither national HPC service initially provided FEA software for HPC users.
- Major third party codes such as Abaqus do not currently scale well on large numbers of processors.
- Many engineers have limited their research to 2D because of compute requirements, both cpu and memory, of large 3D problems.
- Increasing pressure to address this.
- Summer School in ‘HPC in FEA’ jointly run by UoM and National Science Foundation at Manchester in September 2003.

Engineering Areas at Manchester

- Geotechnics:
  - 'Traditional' structural analysis
  - Stochastic analysis
- Biomechanics
  - Medical School
  - Dentistry
- Mechanical Engineering
  - Pressure Vessels
  - Pipe whiplash
- Chemical Engineering
- Earth Sciences
- Aeronautics (CFD)

Engineering codes at Manchester - 1

- Third Party Codes
  - Abaqus
    - Most widely used FEA code – site license.
    - Used on local systems including SGI Origin and IBM SP
    - Jobs typically large memory and small numbers of processors.
    - Problem size limited by memory and scalability
    - Widely used in other UK universities.
  - FLUENT
    - Small number of users, licensed individually.
  - Other software
    - Generally licensed for specific research groups
    - CFX etc.
- Similar at other UK institutions

Engineering Codes at Manchester - 2

- FEA suite of codes written by Prof Ian Smith (Manchester) and Dr Vaughan Griffiths (Colorado)
- Areas covered:
  - Static equilibrium of structures
  - Static equilibrium of linear elastic solids
  - Material nonlinearity
  - Steady State flow
  - Transient problems (uncoupled)
  - Coupled problems
  - Eigenvalue problems
  - Forced Vibrations

Engineering codes at Manchester - 3

- FEA Suite of Codes:
  - About 50 example codes (and 100 library routines).
  - Fortran 90 serial codes used by many engineers at Manchester, and also at many institutions wolrdwide.
  - Element-by-element approach. No matrix assembly.
  - PCG, BiCGStab, Lanczos solver (Harwell library)
  - Low memory, efficient code (matrix operations)
  - Structured or unstructured grids.
  - Problem size limited by cpu and memory of single processor.

Engineering Codes at Manchester - 4

- How do we (computing service) encourage engineers to exploit HPC?
  - Wait for third party packages to scale well?
  - Encourage users to start using alternative parallel software, e.g. PetSC, ScaLAPACK?
  - Provide alternative based on parallelising current codes?

ParaFEM Library - objectives

- Implement highly parallel version of suite of FEA codes.
- Retain code style of serial codes, so engineers can use with little if any knowledge of the parallel coding.
  - Provide both message passing (MPI) and shared memory (OpenMP, MTA) versions.
- Integrate with other packages for mesh generation, preconditioners, alternative solvers and post processing/visualisation.
- Provide framework for engineers to exploit HPC architectures.

Element By Element

- Inherent loop based parallelism throughout code.
- Non-linear and timestepping codes essentially involve loops around the linear solver – thus if linear solver works well, all other codes will (should).
- Stages of codes …
  - Geometric – mesh generation/partitioning
  - Boundary conditions
  - Application of loads
  - Preconditioning (Simple diagonal preconditioner in PCG)
  - Solver (PCG, BiCGStab, Lanczos)
  - Stress recovery
  - Interpretation of results - visualisation

PCG Solver - Serial

```
!--------------------preconditioned c. g. iterations--------------------------
 iters = 0
 iterations  :      DO
              iters=iters+1; u_pp=0._iwp; pmul_pp=.0_iwp
              elements_2 : DO iel = 1, nels
                           g=g_g(:,iel); pmul=p(g)
                           utemp_pp = MATMUL(km,pmul_pp)
                           u_pp(g) = u_pp(g) + utemp_pp
              END DO elements_2
!---------------------------pcg equation solution-------------------------------
              up=DOT_PRODUCT(r_pp,d_pp); alpha= up/ DOT_PRODUCT(p_pp,u_pp)
              xnew_pp = x_pp + p_pp* alpha ; r_pp=r_pp - u_pp*alpha
              d_pp = diag_precon_pp*r_pp; beta=DOT_PRODUCT(r_pp,d_pp)/up
              p_pp=d_pp+p_pp*beta
              CALL checon(xnew_pp,x_pp,tol,converged)
              IF(converged .OR. iters==limit) EXIT
 END DO iterations
 WRITE(11,'(A,I5)')"The number of iterations to convergence was  ",iters
 WRITE(11,'(A,E12.4)')"The central nodal displacement is  :",xnew_pp(1)
```

PCG kernel

- Element–by-element approach dominated by:

```
pmul = p(g)                          ! gather
utemp = MATMUL(km,pmul)              ! matrix-vector
u(g) – u(g) + utemp                  ! Scatter
```

- And vector operations involving (global) dot products

Parallel Implementation

- Partitioning: simple …
  - Elements split across processors
  - Equations spilt across processors
  - Matrix multiplication is local
  - Splits cannot match exactly – nodes (generating equations) are shared by elements which reside on different processors. Could duplicate nodes and update correspondingly, but not done at present.
  - Thus gather and scatter must be performed across processors.
  - Gathering variable amounts of data from different processors. Cannot use simple MPI_GATHER. Could use MPI_GATHERV if appropriate communicators set up. Decided to write our own gather and scatter:

PCG Solver - Parallel

```
!--------------------preconditioned c. g. iterations--------------------------
 iters = 0
 iterations  :      DO
               iters=iters+1; u_pp=0._iwp; pmul_pp=.0_iwp
               CALL gather(p_pp,pmul_pp)
               elements_2 : DO iel = 1, nels_pp
                            utemp_pp(:,iel) = MATMUL(km,pmul_pp(:,iel))
               END DO elements_2
               CALL scatter(u_pp,utemp_pp)
!----------------------------pcg equation solution-------------------------------
               up=DOT_PRODUCT_P(r_pp,d_pp); alpha= up/ DOT_PRODUCT_P(p_pp,u_pp)
               xnew_pp = x_pp + p_pp* alpha ; r_pp=r_pp - u_pp*alpha
               d_pp = diag_precon_pp*r_pp; beta=DOT_PRODUCT_P(r_pp,d_pp)/up
               p_pp=d_pp+p_pp*beta
               CALL checon_par94(xnew_pp,x_pp,tol,converged,neq_pp)
               IF(converged .OR. iters==limit) EXIT
 END DO iterations
 IF(numpe==1)THEN
    WRITE(11,'(A,I5)')"The number of iterations to convergence was  ",iters
    WRITE(11,'(A,E12.4)')"The central nodal displacement is  :",xnew_pp(1)
 END IF
```

Serial -> Parallel

- Call gather()
  - Performs gather for all elements – increasing memory requirements and increasing size of messages, but reducing number of messages
- Matmul
  - For all elements. When element types all the same, stiffness matrix, km is the same, so can perform matrix matrix. In a more general case, km is replaced by storkm(nels_pp,:,:).
- Call scatter()
  - Scatter for all elements

Dot products, convergence criteria etc.

- Different versions of PCG implemented
- Can reduce number of dot products and reduce impact of convergence testing.
- Developments based on paper by Dongarra et al 2004

Typical coding

- Main codes typically about 150 lines – serial or parallel.
- FEA library modules for:
  - Geometry – for different element types
  - Utility
  - …
- Parallel library modules for:
  - Partitioning
  - Gather/scatter
  - …

Generic coding

- Changes made for parallel MPI version, particularly use of gather and scatter routines, can be used in serial and shared memory versions.
- Have run shared memory versions with OpenMP and on MTA (reported at CUG 2003 – MTA particularly suitable for minimising time in gather and scatter).
- Thus single generic main program may be used in any of these environments – user maintains only one version, selecting appropriate library code via f90 USE statement.
- Primary development is for MPI version.

Performance

- Work started on Cray T3D/T3E
- Subsequently most development on SGI Origin/Altix and IBM SP
- Peformance depends on good matrix-vector (or matrix-matrix) and good communiations.
- Original simplistic assumptions about partitioning etc not a problem on best balanced systems (=> Cray!).
- Typically time for gather/scatter is similar to time for matmult, but scales consistently. (Improved versions under development.)
- As communication/computation ratios increases, performance has become more of an issue.

Vector Machines

- What about vector machines – X1?
- Work is dominated by matrix-vector or matrix-matrix, which should work well if vectors are long enough.
- 20 node brick elements generate vectors of length 60 – is this enough?
- Is there enough computation?

Single node performance

- Typical performance on scalar systems. Matrix multiplication (60x60 x 60xnels) about 50% peak performance.
- On X1 SSP:
  - Initial results - about 1% peak!
  - Eventually discovered the problem is the calculation:
    flops = 2.0*ndof*ndof*nels*iters
    (used only in the calculation of a flop count to report performance) . The answer should be about 230GB, but the value returned was about 4GB – 2.0 *maxint
  - By ensuring real arithmetic is used, the correct figure is obtained giving about 80% peak!
  - Note that the matrix multiplication was performed with f90 MATMUL – using BLAS resulted in lower performance (because MATMUL is inlined, avoiding the overheads associated with calling subroutines).

Matrix-matrix/matrix-vector

- Code does do repeated matrix-vector, but X1 recognised that this can be replaced by matrix-matrix, so automatically did so. (Unless it can do matrix-vector at 80% peak!) Not all other compilers do this. On one system, had to use explicit dgemm call for best performance.
- Problems with identical elements (e.g. in biomechanics, use of CT scans can generate voxel elements) can use matrix-matrix, thus achieving very good performance.
- The other extreme with every element different results in matrix-vector computations, potentially with little data re-use.
- Some simple test programs on the X1 indicated that matrix-vector runs about half the performance of matrix-matrix, but this will be very dependent on vector length.
- These provide upper and lower bounds for performance.

Matrix-vector improvements

- Many problems will have some elements the same or at least the same shape and property. This results in duplication, which can be exploited:

Reducing Element Stiffness Storage

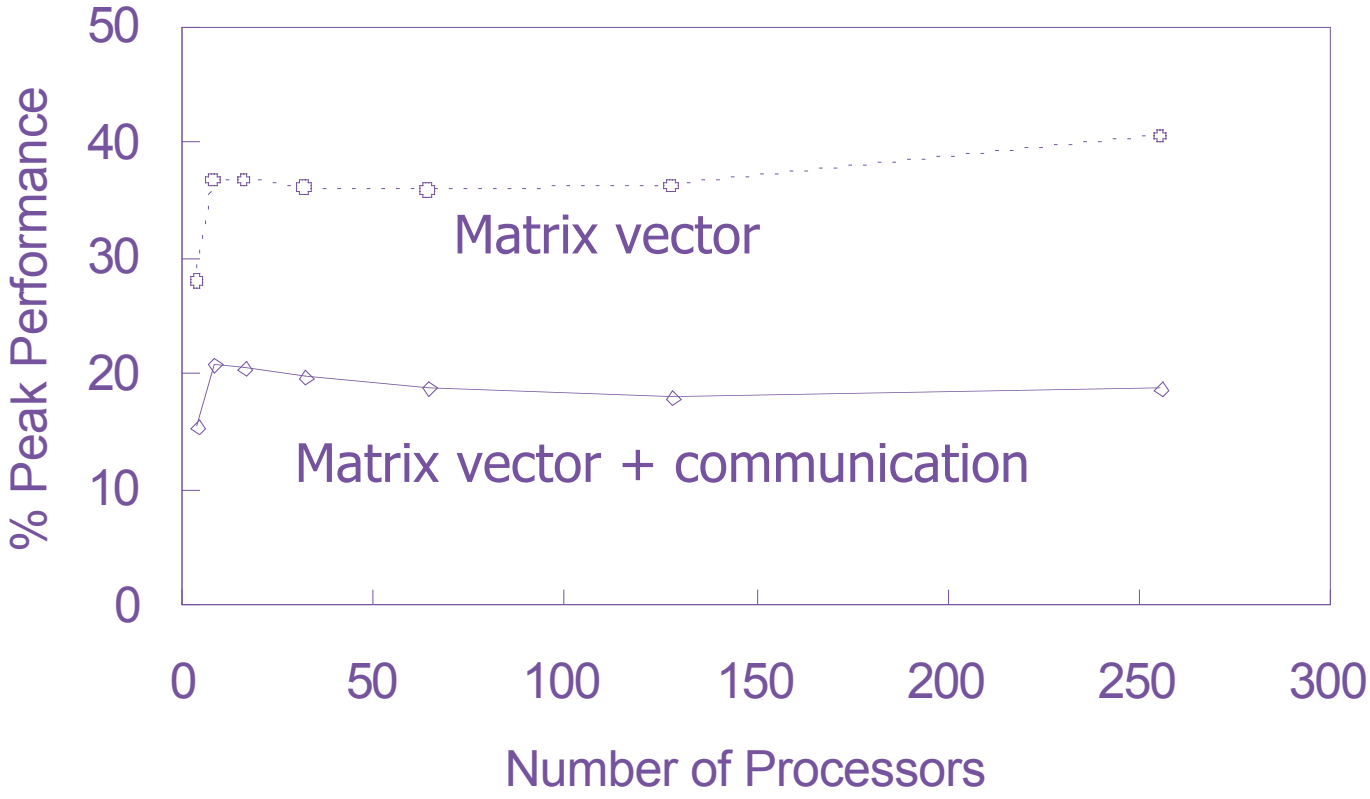
- Consider the full Magnetohydrodyamics stiffness matrix
  - There are 13 unique submatrices for each element
  - Each submatrix has 400 entries

- Break up the element matrix vector computation, replacing

```
do iel=1,nels_pp
   u=matmul(ke,x)
end do
```

```
do iel=1,nels_pp
   u′= matmul ( C11 , x′ )
end do
do iel=1,nels_pp
   u′= matmul ( C55 , x′ )
end do
do iel=1,nels_pp
   u′= matmul ( C15 , x′ )
end do
```

Percentage Peak Performance



Origin 3800 ~300,000 unknowns

Matrix-vector: Superelements

- Can combine elements to generate vectors of length 120, 180 etc.
- Additional computation, but less dense. Higher flop/s but higher flops. Is it worth it?
- Not yet implemented.

Matrix-vector: Coupling

- Coupling different physics at element level
  - Navier Stokes - Pressure + velocity - vector of 68
  - MHD - Pressure + velocity + magnetism - vector of 128
  - Biot - Fluid + solid - vector of 68

Gather/Scatter - scalar systems

- On scalar systems gather/scatter typically takes similar time to matrix multiplication, thus lowering %peak by a factor of 2.
- Looking into ways to reduce this, but it scales, so can still achieve 25% peak across large systems.

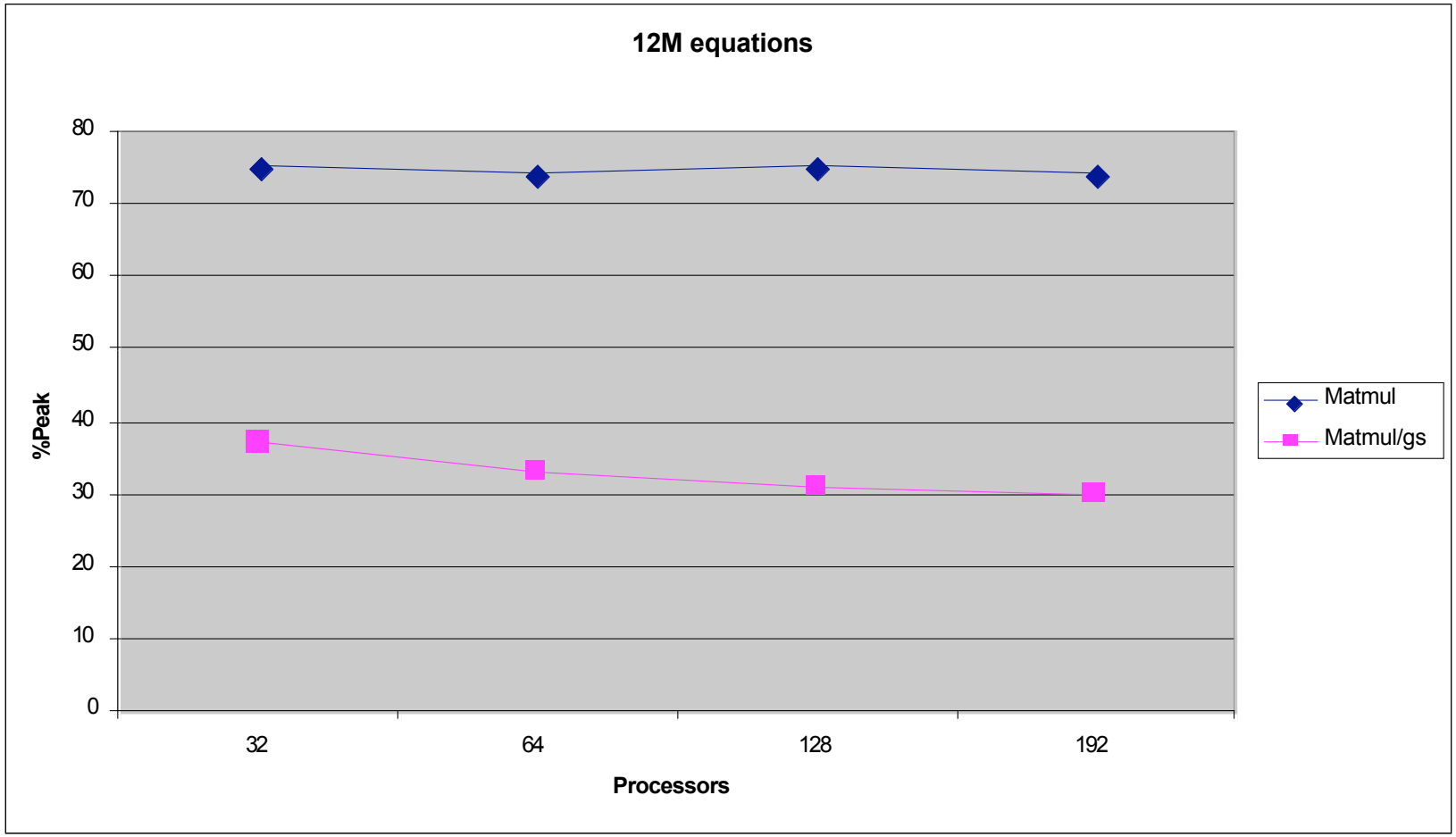
Gather/scatter on X1

- On X1, initially, time in matmul was 25s, and that in the scatter routine 296s!
- However, the time in scatter is dominated by a loop with indirect addressing which was therefore not vectorised.
- There is no recursion in this loop, so the IVDEP directive can be used.
- Time in scatter drops to 34s.
- Still slightly higher percentage of total time than on other systems.
- Uses MPI – currently not optimal on X1. Can try SHMEM or CAF (John Levesque showed simple CAF code for similar scatter).

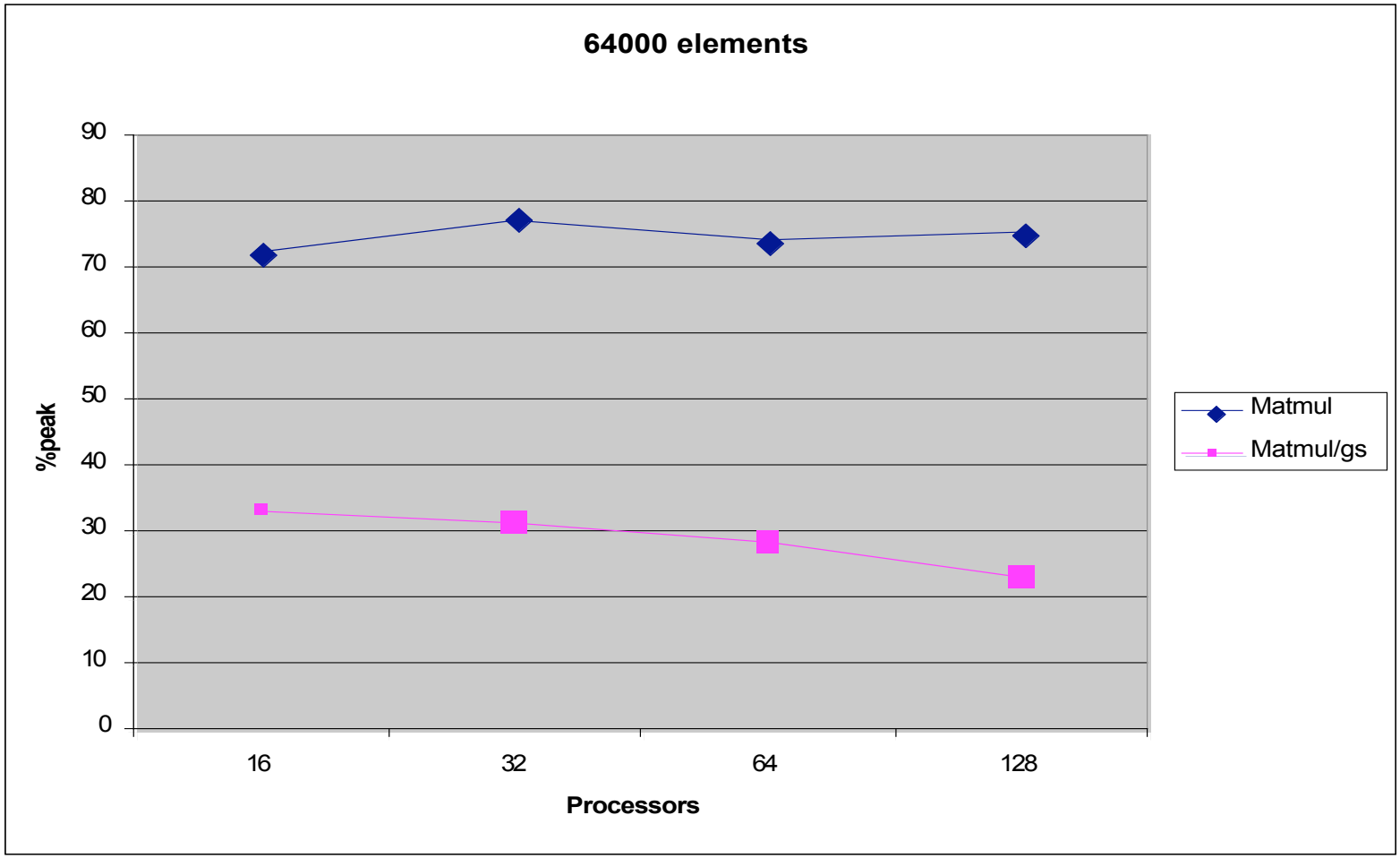
Linear solver – 12M equations

	Total secl	Iters sec	mm sec	mm GF	mm %pk	mm/gs Sec	mm/gs GF	mm/gs %pk
32	298.0	286.8	121.5	77	75%	127.0	38	37%
64	161.4	152.8	61.6	152	74%	75.1	68	33%
128	89.1	81.9	30.5	306	75%	31.9	129	31%
192	63.9	56.7	20.6	453	74%	30.1	184	30%

Linear Solver – 12 M equations



Linear solver – 0.75M equations



MSP Performance

- Matrix-matrix fine:
  - 10GF (~80% peak)
- However gather/scatter takes similar time to SSP, so overall, performance is much lower than on SSP.
- Improvements in communication are key to good performance on MSP.
  - SHMEM of CAF may help, but other changes planned to minimise the communication times likely to be most beneficial.

Other Problem Types?

- Not yet run on X1.
- Given known information from running on other systems, and results so far on X1, expect similar results.

Developments

- Communications
  - Approach adopted by Carey (Texas), uses element-by-element, duplicating nodes on processors. This eliminates communications in gather and communication in scatter is overlapped with computations.
- Provide alternative ‘components’:
  - Mesh partitioning - Par Metis etc
  - Preconditioners
  - Solvers
  - Algebraic multigrid (Adams, Livermore) – excellent performance on very large problems and systems.
  - Visualisation integration
    - Virtual prototyping project

Conclusions

- The ParaFEM software is designed to provide engineers with a framework for solving FEA problems in an HPC environment.
- Previously implemented successfully on scalar MPP systems.
- Easy to port to X1.
  - Requires only the addition of a single compiler directive to obtain good performance on SSP, at least for 20 node brick elements.
  - Vector length is certainly an issue, particularly with simpler element types, but there are ways in which this can be addressed.
- Improvements already planned to improve communications in scalar version will help, particularly with MSP version.
- www.parafem.org.uk

Manchester Computing
Supercomputing, Visualization & eScience

SVE @ Manchester Computing

World Leading Supercomputing
Service, Support and Research

Bringing Science and Supercomputers Together

www.man.ac.uk/sve
sve@man.ac.uk

THE UNIVERSITY of MANCHESTER